Cake Cutting Algorithm Project Report

Bangxi Yu, Jing-Wu Chen
Repo name: happyschduling

I. Description

The main idea of our project is to implement algorithms to divide and assign working time slots for employees fairly. We try to solve this problem in two ways, one is for each employee, he/she picks the time slots he/she prefers to work on, the other is employee can pick the time slots he/she does not like. We start solving the problem from the simple situation, there are only two employees. Then we try to solve the more complex situation, there are more than 2 employees.

II. Main Idea with cake cutting and fair division

In our project, we treat the whole working time slots as a cake, then we can use the algorithms we learn from the class, and do some adaptations to make the algorithms work on our project. To make fair division, we need to know everyone's preference, since the working time slots are indivisible items, we give some points to employees, so that they assign the points on the time slots they prefer, and then we can make our decision based on their preferences. Another thing we do is, although the time slots are indivisible items, we try to have a test on whether we can treat the time slots as divisible items if we make the time slots small enough.

Our target is to make everyone satisfied with the time slots he/she gets, the way we define "satisfied" is based on the points everyone has (we assume it is p), the points everyone receives (we assume it is q) and the number of employees (we assume it is n). To every employee, when he/she submits his/her preference, he/she will have a value on every time slot (everyone will have points to assign on his/her preferred time slots). After division, each employee will get some time slots (these time slots are worth q to him/her). If $q >= p/n$, we say that this employee is satisfied.

III. Algorithms and Modeling

We implement five algorithms in our project to solve the problem. Our time period unit is one month by default (one day is one time slot, 30 time slots per month), and each employee will have points (1000 by default) to assign on 30 days.

The first two algorithms is for $n = 2$, there are only two employees.

1) Pick By Turn(Alternation)

We have two preference files of two employees(p1 and p2), according to the preference files, WLOG, we assume A is p1 picks then p2 picks, B is 2 picks then p1 picks. Then p1 and p2 pick the time slots by A-B-A-B-A-B-A-B-A-B-A-B-A-B-A turn, each time, the employee will pick the highest valued slots from available time slots. At the end of division, each employee will receive 15 days.

2) Divide and Choose

Still for n = 2, employee p1 and p2. WLOG, we divide the time slots into two parts according to p1's preference, then let p2 pick first. In our project, there are two issues for this algorithm. One issue is that we use points to assign on time slots, it is highly possible that we cannot divide time slots into two parts, and either one is worth exact 0.5 of the total points to p1, so we try to make each part of time slots is close to 0.5. The other issue is that in traditional divide and choose algorithm, p1 uses one cut to divide the cake into two pieces, but since our project is about working time or rest time, if we keep doing this, the result of this will be strange. For example, p1 will work on the first ten time slots, and p2 will work on the rest, this does not meet the common sense. What we do here is to shuffle the time slots before p1 cuts, it is the same as we divide the whole cake into many pieces then make them together in arbitrary order to get a reconstructed cake.

The other three algorithms is for n>2, there are n employees.

3) Moving knife

The strategy here is the same as normal moving knife, when the value to the left of "knife" is worth p(i)/n to employee(i), employee(i) will call "stop", then get the time slots to left of "knife". The problem here is similar with divide and choose, so we also do a shuffle here to make a reconstructed "cake". The reason we implement this algorithm here, is to test that when we make the time slots small enough, can we use the algorithms for divisible items to solve this problem.

4) Sealed bids

We use points instead of money to do the sealed bids. We compute the fare share, surplus, difference and settlement for each employee with points. The problem here is, in normal sealed bids, the employee who received more than he/she should get will give out some money as remedy, but we use points to bid so we cannot do that. We try to do some adjustments to make sealed bids can work on this problem, we assume that each employee has a points bank, at the beginning of each month, employer will put a constant number of points (1000 by default) into each employee's account, and each month employees can take out some of the points from their point-bank to bid the time slots they want, the winner will get the time slots, but

need to pay some points to the loser. In really life, the employer even can have some incentives, like late and leave early will lose some points and working hard will earn some points.

5) Modified Adjusted Winner

The basic idea is similar to Adjusted Winner, but some modification is made. Preferences should be submitted in advance, 1000 points for each employee. If the point on one time slot is zero, it means the employee has no desire, or cannot work on that time slot. If one employee does not particularly want to, but does not mind to work on one time slot, 1 point should be given for that time slot.

At the beginning, we deal with time slots that are not competed. In other words, time slots that only one of the employees wants will be allocated first. Then, bids start, the employee with highest value gets the time slot. The next step is to transfer some of the time slots from winners (get a lots time slots and happy) to losers (get a few time slots and unhappy) to make the allocation fair. First, each employee's happiness is calculated by the amount of points they received currently, transfers are made based on that. If the difference between employees' happiness is greater than the threshold (200 by default), the algorithm will try to balance it. Also, we do not want inefficient transfers, what we do here is, transfer will never be made if one side of the transfers lost great amount of points and the other side only receives a little (idea is similar with computing ratio in normal adjusted winner). In our project, we set the ratio to 1.8 by default. At the end, the days nobody bids on will remain unallocated. We will not force any employee to take the day they don't want in this algorithm. In real life, if we are pick working time slots, the unallocated remaining days will be left to the manager/employer.

IV. Analysis method

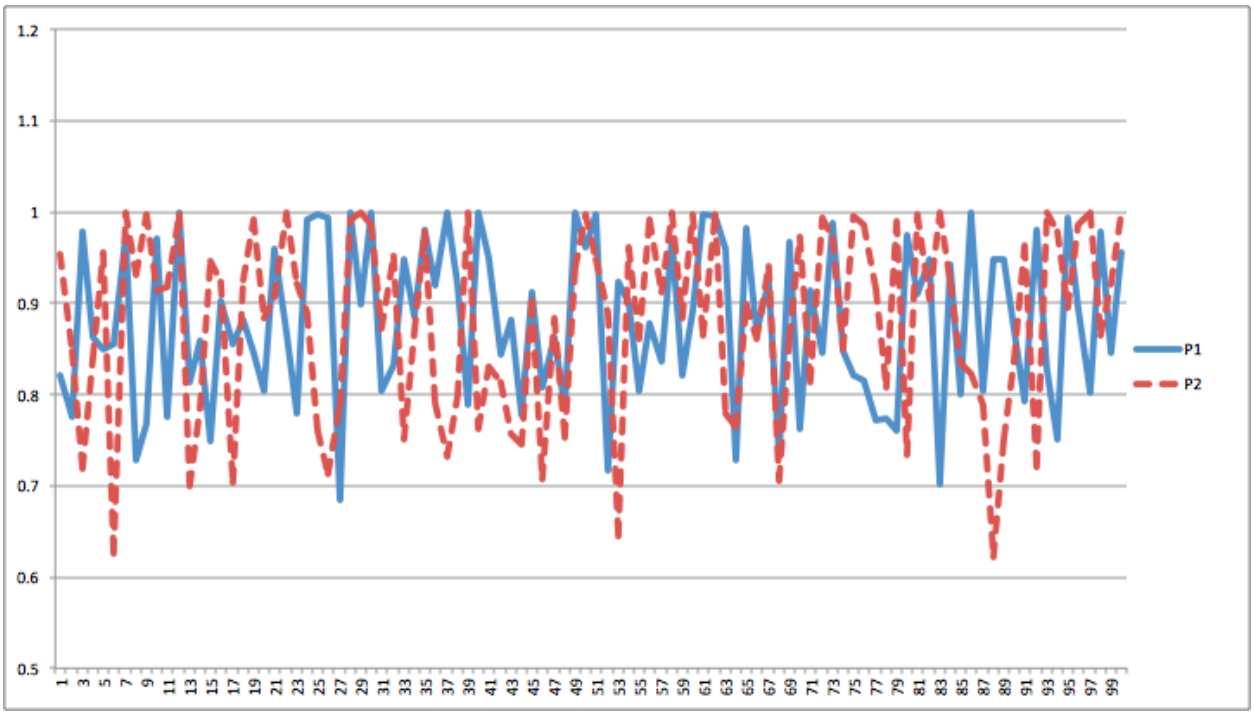
The main analysis method we use is simulation**.**

The most important factor to the result is the preference file, we use three ways to simulate the preference file: 1) Use the random.randint() function to randomly generate the preference file; 2) Make the preference file with extreme situations by hand (like all points for one day); 3) Let other people to make their own preference file for us. We run our each algorithm with various preference file for many times and compare the result.

Another parameter for our simulation is the whole time slots. As we mentioned above, in D&C and Moving Knife, we try to treat the time slots as divisible items. In order to make the result meet the common sense, we use random.shuffle() function to shuffle the whole "time slots" each time before division.

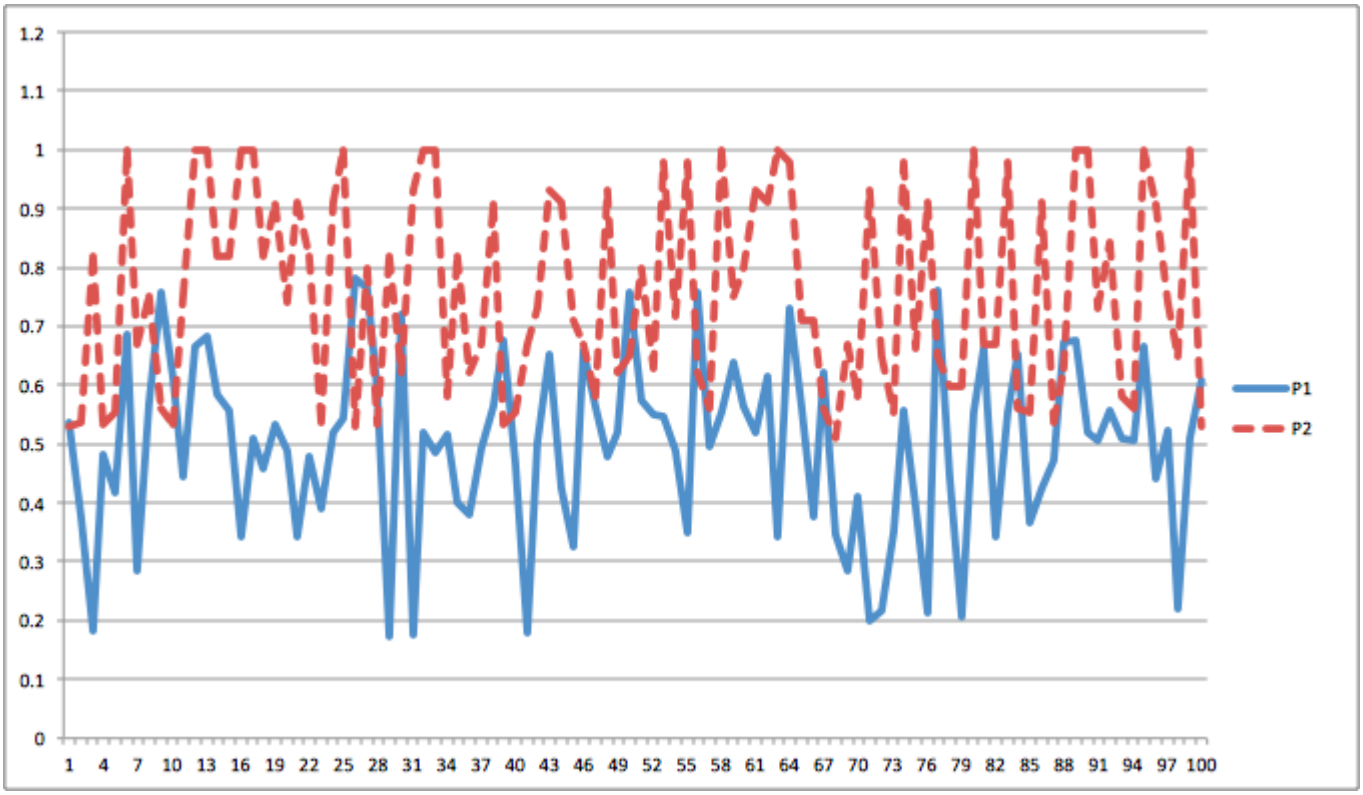
V. Result and Analysis

1) Below is the result graph for pick by turn of running 100 times with different preference file. The Y-axis is the value each employee gets (the same for all graphs). From the graph we can easily find that, for n = 2 people, when the number of time slots is much bigger than n (here the number is 30), Pick by Turn works very well. We think the reason is that, since the number of time slots is much bigger than n, there will be only a small overlap for the two employees, so they are easy to get the time slots they want. But the worst situation is, for example, p1 and p2 like one same time slot and both put a lots points on it, so one of them will be disappoint, Pick by Turn cannot avoid or deal with this worst situation.

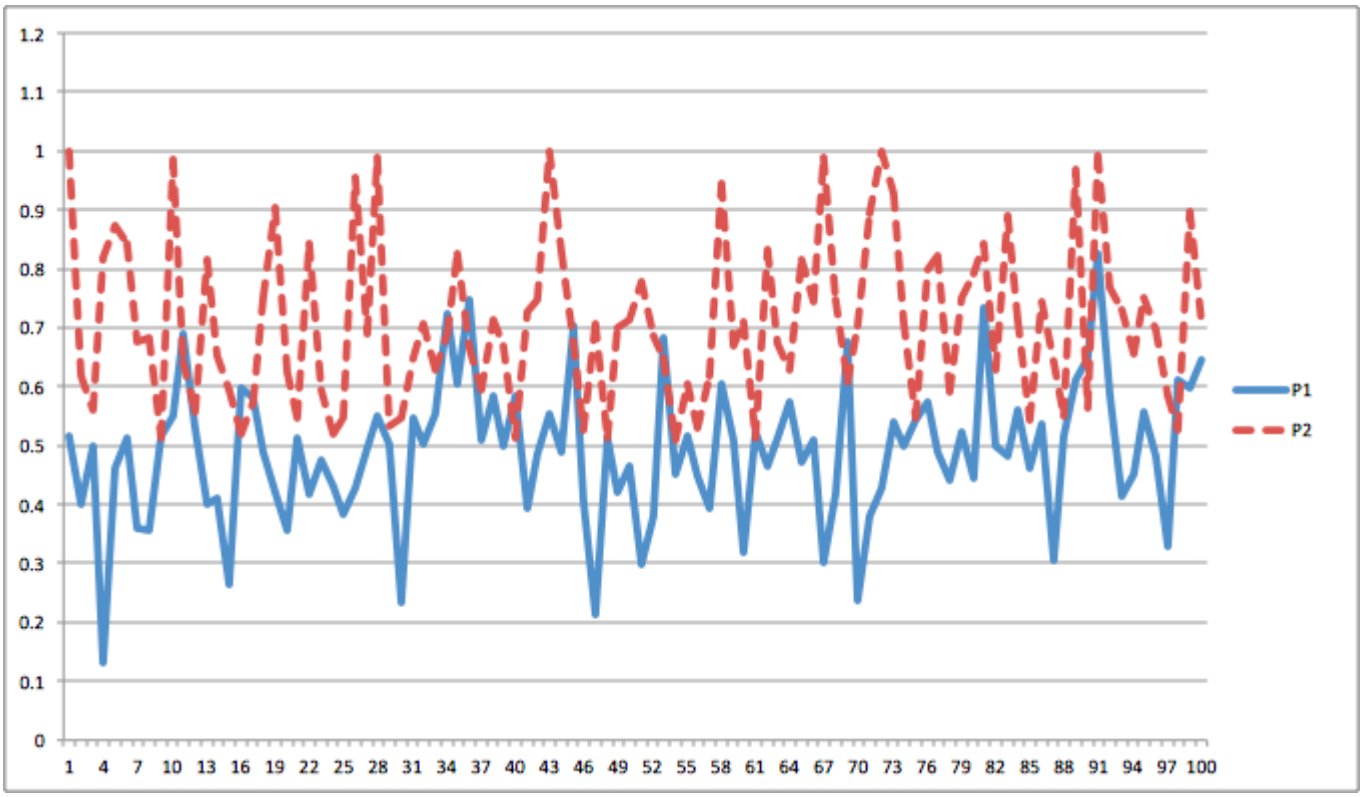


a). Pick by Turn result of running 100 times with different preference

2) Below are two graphs((b) and (c)) of Divide and Choose (D&C) result, the first one we use same preference files with shuffled time slots to run 100 times, the second one we use different preference files with shuffled time slots to run 100 times. In the D&C procedure, p1 will cut the whole “time slots” into two parts, then p2 will choose first. From the result we can see, p2 will always get more than 0.5, but sometimes p1 even only get about 0.2. The reason is time slots are indivisible items, if p1 put a large number of points on one time slots, sometimes it might be hard for p1 to divide the whole time slots into two parts equally. We also try to make each time slot smaller (for example, we set 4 time slots per day, so that there are 120 time slots per month), but we still cannot avoid this problem.

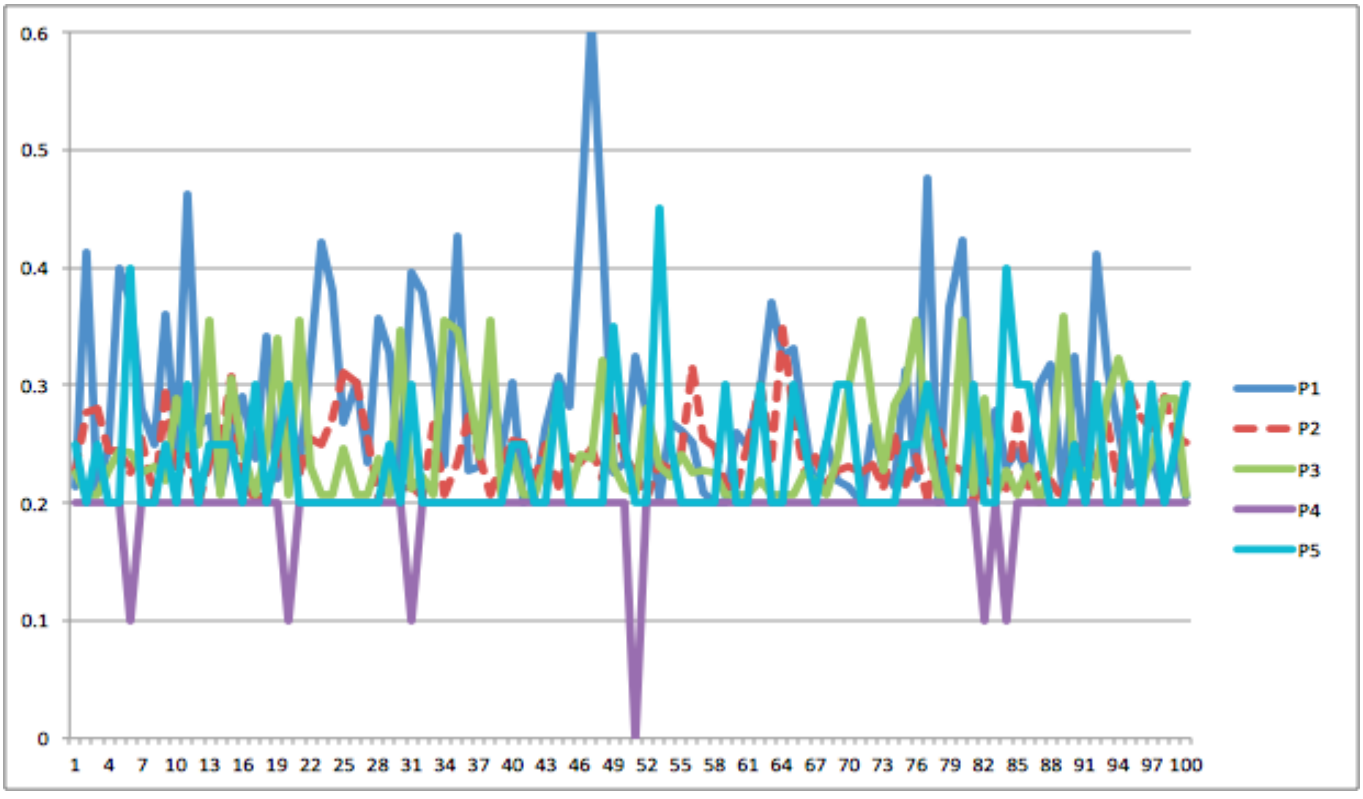


b). Divide and Choose result of running 100 times with same preference file

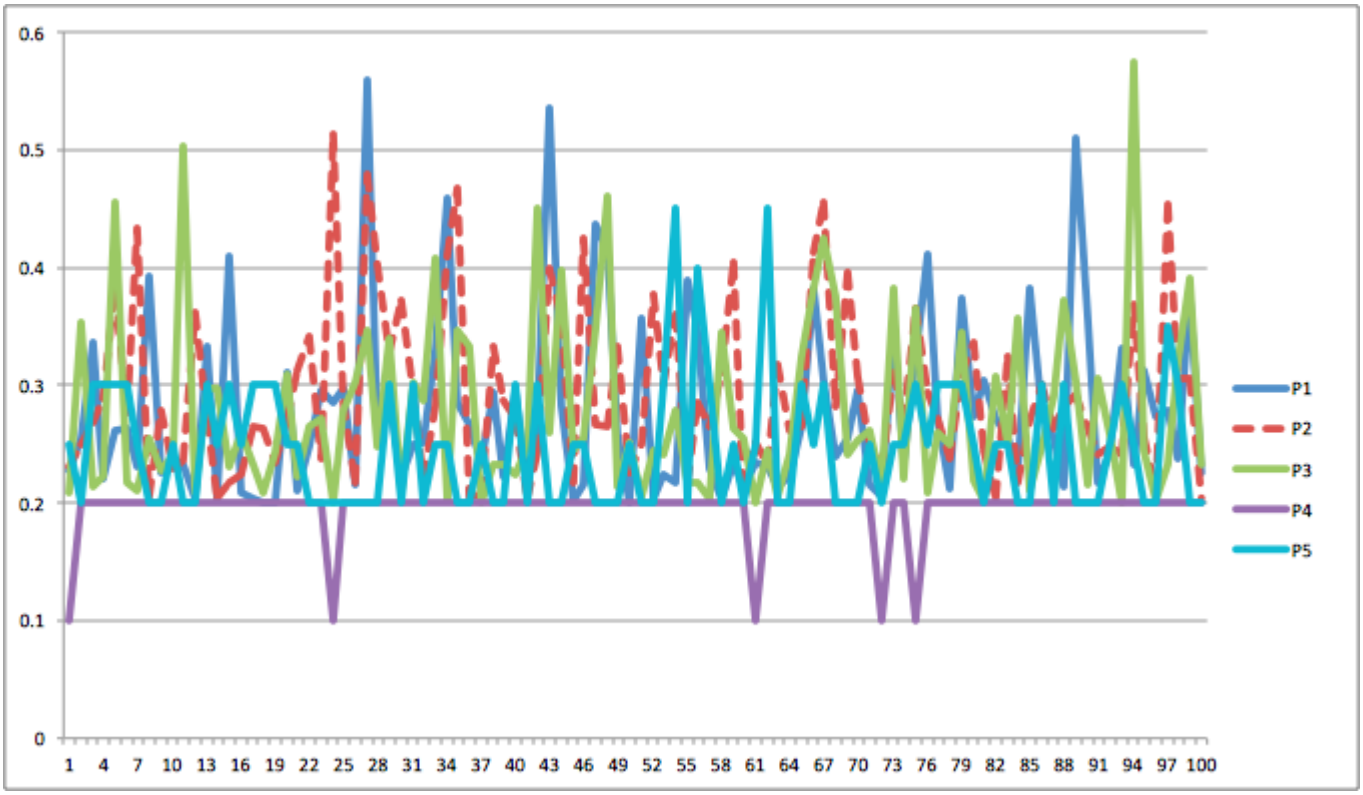


c). Divide and Choose result of running 100 times with different preference file

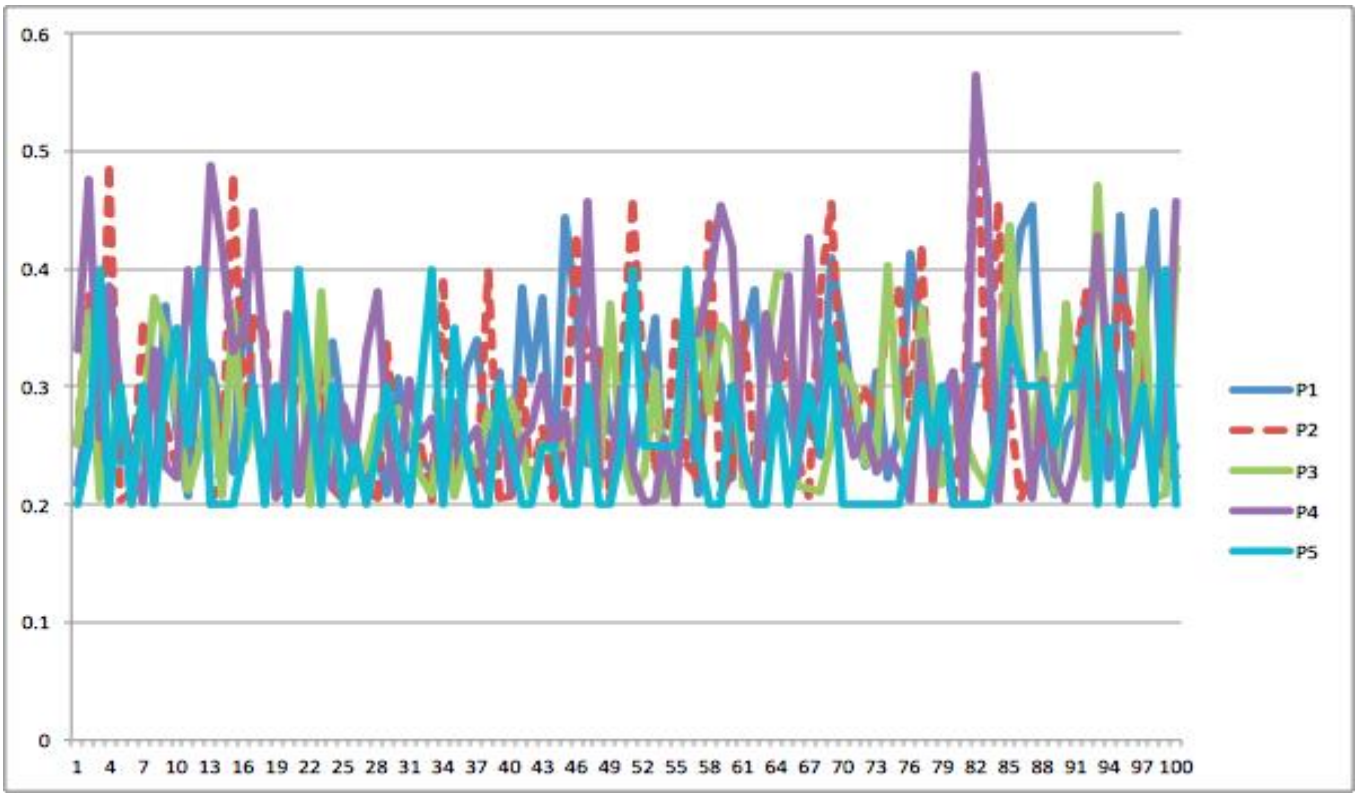
3) Below are results for Moving Knife for n = 5, as D&C, in graph (d) we keep the preference file same, and in graph (e) we use different preference file (we hack the p4's preference file by hand to make an extreme situation that p4 put most points on one day). But in first two graphs, there is an obvious drawback of this algorithm, when more than one employee call "stop" at the same time, we always give the time slots to the employee with small number, for example, if p1 and p3 call "stop" at the same time, p1 will get the time slots. Then we do some adjustments to our algorithm to deal with the "stop conflict", when more than one employee call "stop" at the same time, we will compare the value of selected time slots (left to the "knife") for each of them, then give the selected time slots to the employee with higher value. Graph (f) is the result for the improved algorithm, we can see that, now every employee can get more than p/n value (here is 0.2) this time.



d). Moving knife result of 100 times with same preference file



e). Moving knife result of 100 times with different preference file



f). improved Moving knife result of 100 times with same preference file

4) For sealed-bids, we run the algorithm for 12 months with n =5, start with 1000 points, and each month employees will get 1000 points plus the points lost/won last month to bid. From the result table (only show the result of 6 months here) we can see that, no one will always win (get a lots days) or lose (get a few days). From the table we also can find that the sum of days each employee gets is close. The extreme situation for this algorithm is that one employee may get 0 day in some month. And what we can do to make the simulation better is, instead of randomly generating, try to record the preference file, for example, one employee likes Sunday, so he/she will always put more points on Sunday.

Table). Sealed-bids for 6 continuous months

		Month1	Month2	Month3	Month4	Month5	Month6	total
P1	Get(days)	6	4	5	4	8	8	35
	pay	67	-126	-67	274	-45	284	
P2	Get	7	6	5	5	3	5	31
	Pay	82	23	-45	4	-82	242	
P3	Get	6	6	8	6	12	6	44
	Pay	14	214	-208	294	-99	-33	
P4	Get	5	9	7	5	4	6	36
	Pay	-260	-19	112	-378	32	-283	
P5	Get	6	5	5	10	3	5	34
	Pay	95	-305	205	-196	194	-213	

5) The graph below shows the test result of Modified Adjusted Winner running over 100 times with random generated data. Over 100 tests, only in case 53, one of the employees' (p1) satisfactory rate falls below 0.4. In case 53, before adjustment made, the satisfactory rate is as below:

p1= 0.313 p2= 0.921 p3= 0.545 p4= 0.939 p5= 0.896

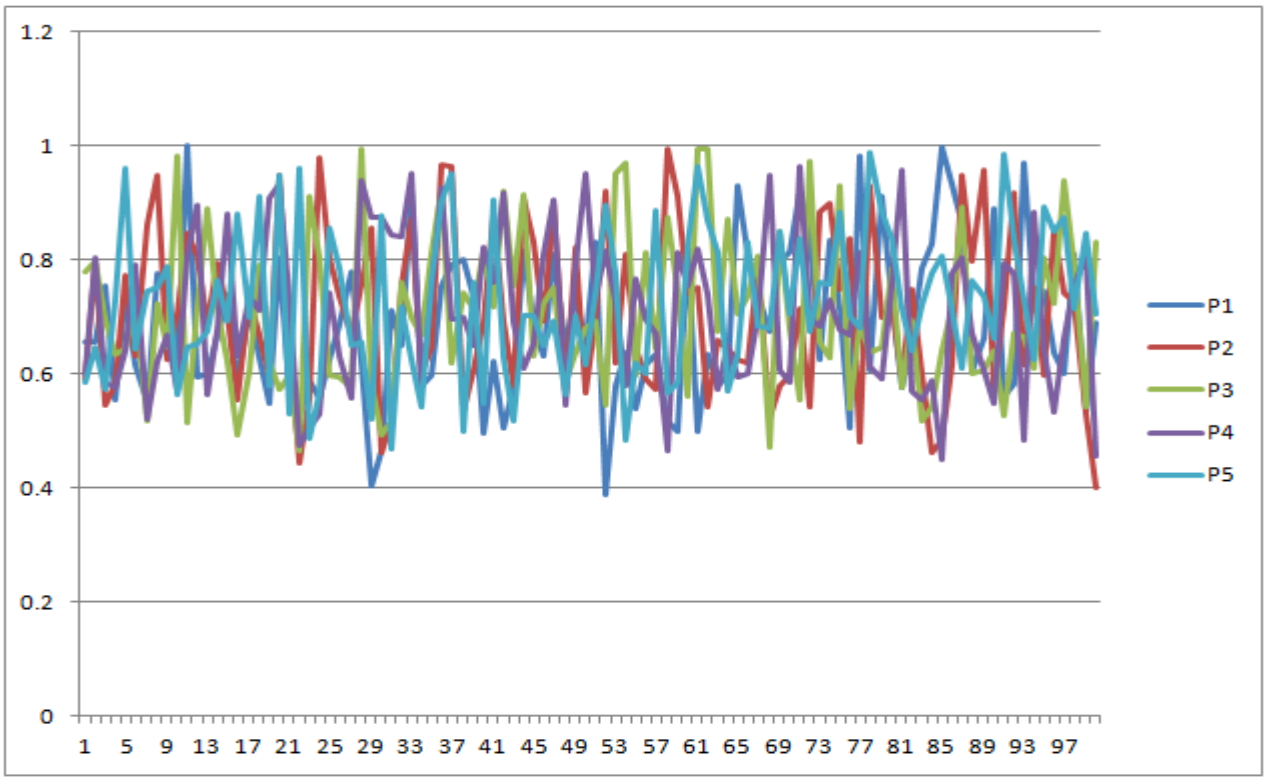
After adjustment, one day is transfer from p4 to p1, as result the satisfactory rate become as the following:

p1= 0.389 p2= 0.921 p3= 0.545 p4= 0.816 p5= 0.896

After examining the preference data, we discover that if we raise the threshold of the ratio that transfer can be made by 0.027 (the original setting is 1.8), the 16th of the month will be transferred from p5 to p1. Consequently, the satisfactory rate of p1 and p5 would be 0.488 and 0.715 respectively. Also, we can go further, if we raise the threshold by 0.056 (to 1.856), one more day will be transferred from p2 to p1. That makes the satisfactory rate become as below:

p1= 0.592 p2= 0.728 p3= 0.545 p4= 0.816 p5= 0.715

From the above, we can get better result if we find an optimal setting.



g). modified Moving knife result of 100 times with different preference file

VI. Conclusions

For the simple n = 2 situation.

Since the number of time slots is more than n, the Pick by Turn (Alternation) algorithm is a good solution. In most time, there will only be a small overlap, so both two employee can get the time slots they want. And Pick by Turn algorithm can work on both working time slots and rest time slots picking, and can set the number of days each one will pick.

The Divide and Choose (D&C) algorithm does not work well in this problem, although we can make the time slots smaller, there will be highly possible than the cutter only can divide the whole time slots into two parts, one is bigger than 0.5, one is less than 0.5, and after chooser picked, cutter received the less than 0.5 part.

For the complex n > 2 situation.

The improved Moving Knife is a solution. We treat the time slots as divisible items, and from the test, it seems that the algorithm works well. The only thing need to care is how to deal with the conflict, since no matter how small each time slot is, we still may meet the situation that when more than one employee call “stop” at the same time, but they have different value on the selected part. We always give the selected part to the employee who has higher value on it. This algorithm is also can work on both working time slots and rest time slots picking. In our project, the indivisible items (here are time slots), can be very smaller, and the number of items is more than the number of player, so we can use the algorithm on divisible items to solve the problem.

The Sealed-bids is another solution, we run the algorithm for 12 months (we only show the result of first 6 months above the result of 12 months is saved in our repo), and nobody will dominate the bid. From the result of 6 months we show above, the number of days each employee gets is close, if we can run the algorithm for a long time, the number will be closer. This algorithm is better to be used on picking rest time slots, since there may be some employees get nothing in one month, it does not meet the common sense. When we use this algorithm to pick the rest time slots, we also can set the number of days each employee can get.

From the tests, we find Modified Adjusted Winner has better performance than Moving Knife. In our test, for n=5, each party can receive at least 400 points in most of the cases. The worst case in our test is 389 points which is only slightly below 400 points. Each employee can get a certain amount of points (around 600-800, at least 400 for n =5), but the number of days each can get is unfixed. Also, the days which nobody wants will remain unallocated, so this algorithm is not efficient. If we want to use this algorithm for employees to pick working time slots, additional settlement is still needed after running this algorithm. If we want to assign rest time slots, this algorithm will be enough.